Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Разработка статического сервера»

Студент ИУ7-73Б (Группа) _____ (Подпись, дата) Т. А. Мусин (И. О. Фамилия)

Руководитель курсовой работы _____ (Подпись, дата) М. Н. Клочков (И. О. Фамилия)

2024 г.

РЕФЕРАТ

Отчет n с., m рис., o табл., p источн., q прил.

Целью данной работы является реализация статического сервера для отдачи файлов с диска на основе пула потоков и мультиплексора.

В работе проведен анализ предметной области, в котором рассмотрены основные характеристики протокола HTTP, преимущества использования пула потоков и принцип работы мультиплексора poll(). Спроектирован алгоритм обработки сообщений от клиента и реализован статический сервер.

Проведено нагрузочное тестирование реализованного сервера для сравнения производительности его работы с сервером Nginx.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Статический сервер — это сервер, который обслуживает статические файлы, такие как HTML, CSS, JavaScript, изображения, шрифты, видео и другие ресурсы, без какой-либо динамической обработки на стороне сервера. Его основная задача — принимать запросы от клиента (например, веб-браузера) и возвращать запрашиваемые файлы, как они есть, из определённой директории.

Запросы всегда возвращают заранее подготовленные файлы, поэтому поведение статического сервера легко прогнозируется. Статические серверы не поддерживают обработку бизнес-логики, взаимодействие с базами данных или создание страниц «на лету». Для этих задач используются динамические серверы.

Цель работы — реализовать статический сервер для отдачи файлов с диска на основе пуля потоков и мультиплексора poll.

Чтобы достичь поставленной цели, требуется решить следующие задачи:

— провести анализ предметной области,

— спроектировать алгоритмы работы сервера,

— реализовать программу,

— провести нагрузочное тестирование.

1 Аналитическая часть

1.1 HTTP

HTTP (HyperText Transfer Protocol) — это протокол прикладного уровня, используемый для передачи данных в распределенных информационных системах, таких как Всемирная паутина (World Wide Web). Он является основой взаимодействия между клиентами (например, браузерами) и серверами, обеспечивая обмен гипертекстом, мультимедийным контентом, данными и другими ресурсами [1].

Основными характеристиками протокола HTTP являются:

— Клиент-серверная архитектура: HTTP основывается на модели клиент-сервер: клиент (обычно браузер) отправляет запрос серверу, а сервер обрабатывает его и отправляет ответ.

— Без состояния (stateless): каждое взаимодействие между клиентом и сервером в HTTP независимое и не сохраняет информацию о предыдущих запросах. Это упрощает протокол, но требует дополнительных механизмов (например, cookies или session tokens) для реализации сессий.

— Текстовый протокол: HTTP-запросы и ответы обычно представлены в текстовом формате, что делает их легкими для анализа и отладки.

— Простота использования: HTTP предоставляет четкую и понятную структуру запросов и ответов, что упрощает интеграцию в различные приложения.

HTTP-запрос состоит из следующих частей:

1. стартовая строка,
2. заголовки,
3. тело запроса.

Токен метода, находящийся в стартовой строке, указывает действие, которое должно быть выполнено над ресурсом, идентифицированным с помощью URI. Токен метода может быть: GET, POST, PUT, DELETE, HEAD, OPTIONS, PATCH и другие [2].

В листинге 1 приведен пример HTTP запроса с использованием метода POST.

Листинг 1 – пример HTTP-запроса

```
POST /users HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 50

name=FirstName%20LastName&email=bsmth%40example.com
```

1.2 Пул потоков

Пул потоков (thread pool) — это механизм управления потоками в многопоточных приложениях, представляющий собой заранее созданный набор потоков, которые могут переиспользоваться для выполнения задач. Он позволяет эффективно управлять созданием, использованием и завершением потоков, минимизируя накладные расходы, связанные с их частым созданием и уничтожением [3].

Использование многопоточности является стандартным методом, используемым для обработки асинхронных запросов, с которыми обычно работают веб-серверы и серверы баз данных. Для таких приложений многопоточность может повысить время ответа сервера, масштабируемость и пропускную способность, а также улучшить взаимодействие между процессами. Из-за постоянного потока сетевых запросов веб-серверы и сетевое промежуточное программное обеспечение особенно часто попадают в категорию приложений, требующих большое количество операций ввода-вывода. Подавляющее большинство их времени тратится на ожидание ввода-вывода [4].

1.3 Poll

Системный вызов poll() предоставляет программе механизм мультиплексирования ввода/вывода по набору файловых дескрипторов.

В листинге 2 представлена сигнатура функции poll().

Листинг 2 – Сигнатура функции poll

```
#include <poll.h>

int poll(struct pollfd fds[], nfds_t nfds, int timeout);
```

Параметры функции poll():

1. fds — массив элементов типа struct pollfd, который содержит файловые дескрипторы, которые будут отслеживаться с помощью функции poll();

2. nfds — количество элементов в массиве fds;

3. timeout — верхний предел времени, на которое будет блокироваться функция poll().

В листинге 3 преведено определение структуры pollfd.

Листинг 3 – Определение структуры pollfd

```
1 struct pollfd {
2     int fd;
3     short events;
4     short revents;
5 };
```

Поля events и revents структуры pollfd являются битовыми масками. Вызывающий объект инициализирует события, чтобы указать события, которые будут отслеживаться для файлового дескриптора fd. При возврате из функции poll() значение events устанавливается таким образом, чтобы указывать, какое из этих событий действительно произошло для данного файлового дескриптора [5].

Вывод

В данном разделе были рассмотрены основные принципы работы протокола HTTP, его характеристики, методы и пример запроса. Определены основные преимущества использования пула потоков для веб серверов, в том числе тех, что отдают статический контент. Рассмотрен системный вызов poll(), его сигнатура и принцип работы.

2 Конструкторская часть

2.1 Алгоритм обработки сообщений от клиентов

На рисунке 1 представлен алгоритм обработки сообщений от клиентов.

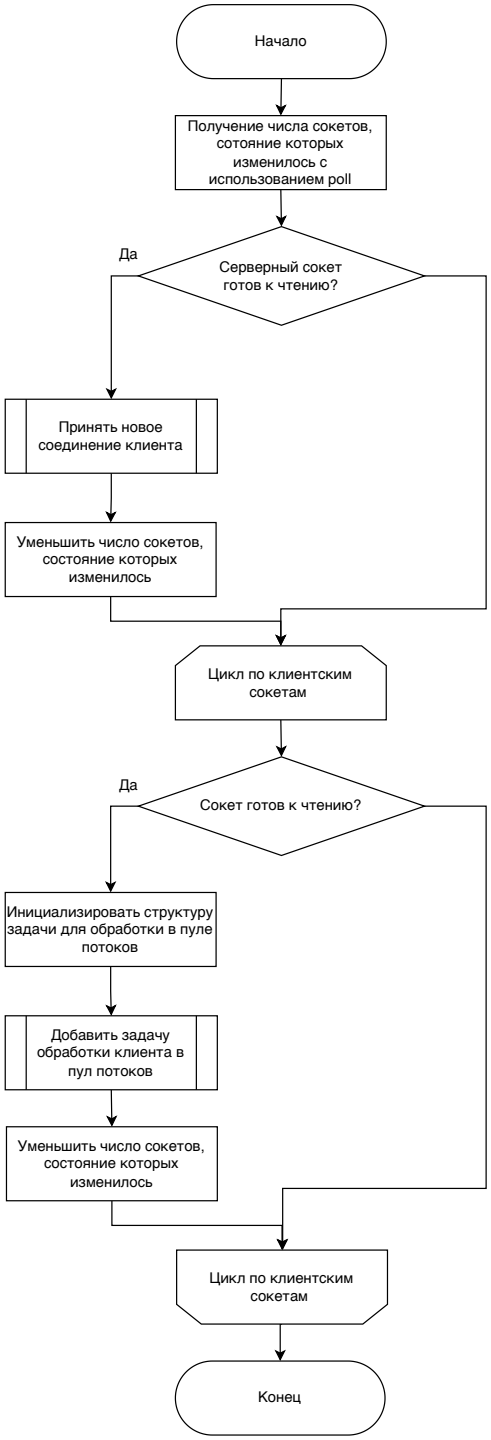

Рисунок 1 – Алгоритм обработки сообщений от клиентов

Данный алгоритм обрабатывает соединения с клиентами в серверном приложении. Системный вызов poll() возвращает количество сокетов, которые

должны быть обработаны. В первую очередь проверяется серверный сокет, для проверки необходимости создания нового сокета для подключения клиента к серверу. Затем обходятся все клиентские сокеты в поиске тех, на которые пришли сообщения. Как только сокет, готовый к чтению находится, задача его обработки помещается в пул потоков.

Вывод

В данном разделе был спроектирован алгоритм обработки сообщений от клиентов. Описана последовательность чтения сокетов клиентов и сервера и дальнейшая обработка сообщения клиента в пуле потоков.

3 Технологическая часть

3.1 Средства реализации

Для реализации статического сервера для отдачи файлов с диска был выбран язык программирования C, так как данный язык позволяет решить поставленную задачу реализации сервера с использованием мультиплексора poll и пула потоков.

3.2 Реализация алгоритма обработки сообщений от клиентов

В листинге 1 представлена реализация алгортима обработки сообщений от клиентов.

Листинг 1 – Реализация алгоритма обработки сообщений от клиентов

```
1  int wait_client(server_t *server) {
2    server->clients[0].fd = server->listen_sock;
3    server->clients[0].events = POLLIN;
4    int numfds = 0, maxcl = 0;
5    int first = 0;
6    while (1) {
7      numfds = poll(server->clients, maxcl + 1, 5000);
8      if (numfds < 0) {
9        LOG_ERROR("poll error");
10       continue;
11     }
12     if (server->clients[0].revents & POLLIN) {
13       int client_sock = accept(server->listen_sock, NULL, NULL);
14       if (client_sock < 0) {
15         continue;
16       }
17       long i = 0;
18       for (i = 1; i < server->cl_num; ++i) {
19         if (server->clients[i].fd < 0) {
20           server->clients[i].fd = client_sock;
21           server->clients[i].events = POLLIN;
22           break;
23         }
24       }
25       if (i == server->cl_num) {
26         LOG_ERROR("too many connections");
27         continue;
28       }
29
```

```
30        if (i > maxcl) {
31          maxcl = i;
32          LOG_INFO("Max clients: %d", maxcl);
33        }
34        if (--numfds <= 0) {
35          continue;
36        }
37      }
38      for (int i = 1; i <= maxcl; ++i) {
39        if (server->clients[i].fd >= 0 &&
40            server->clients[i].revents & (POLLIN | POLLERR)) {
41          worker_sock_t worker_sock;
42          worker_sock.clientfd = &server->clients[i].fd;
43          worker_sock.wd = server->wd;
44
45          tpool_add_work(server->pool, worker, &worker_sock);
46
47          if (--numfds < 0) {
48            break;
49          }
50        }
51      }
52      tpool_wait(server->pool);
53    }
54  }
```

3.3 Демонстрация работы программы

В листинге 2 продемонстрирована корректная работа разработанного сервера. На данном примере сервер прослушивает соединения на порте 9990. Для получения ответа делается GET запрос по пути к index.html файлу.

```
timurmusin@Timurs-MacBook-Air ~> curl -X GET http://localhost:9990/Users/timurmusin/BMSTU/cn/coursework/ics7-cn-cw/src/build/static/index.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="description" content="An introduction to computer networks and their key components">
    <title>Computer Networks</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            line-height: 1.6;
            margin: 0;
            padding: 0;
            background-color: #f4f4f9;
            color: #333;
        }
        header {
            background: #0073e6;
            color: white;
            padding: 10px 20px;
            text-align: center;
        }
        nav {
            display: flex;
            justify-content: center;
            background: #004080;
            padding: 10px;
        }
        nav a {
            color: white;
            text-decoration: none;
            margin: 0 15px;
        }
        nav a:hover {
            text-decoration: underline;
        }
        main {
            padding: 20px;
        }
        section {
            margin-bottom: 20px;
        }
        footer {
            text-align: center;
            background: #004080;
            color: white;
            padding: 10px 0;
            position: relative;
            bottom: 0;
            width: 100%;
        }
    </style>
```

Рисунок 2 – Демонстрация работы реализованного сервера

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени, представлены далее.

— Процессор: Apple M2 [6].

— Оперативная память: 16 ГБайт.

— Операционная система: MacOS Monterey 12.5.

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.2 Постановка исследования

Целью данного исследования является проведение сравнение результатов прохождения нагрузочного тестирования с использованием утилиты Apache Benchmark (ab) [7] с сервером Nginx [8].

Исследование направлено на изучение поведения разработанного сервера под нагрузкой и сравнение его производительности с Nginx.

В рамках тестирования был использован HTTP-запрос типа GET, направленный на URI, указывающий на файл index.html, который представляет собой простой статический ресурс. Такое тестирование позволяет оценить скорость обработки запросов, устойчивость сервера при высоких нагрузках и эффективность его работы в типичных сценариях использования.

4.3 Результаты исследования

В таблице 1 представлены результаты проведения нагрузочного тестирования разработанного сервера и сервера Nginx для 10 клиентов.

Таблица 1 – Время прохождения нагрузочного тестирования для 10 клиентов

Количество запросов	Время прохождения тестирования разработанного сервера, с	Время прохождения тестирования сервера Nginx, с
100	0.035	0.030
500	0.086	0.082
1000	0.148	0.127
2000	0.291	0.215
3000	0.416	0.274
4000	0.537	0.405
5000	0.678	0.453

В таблице 2 представлены результаты проведения нагрузочного тестирования разработанного сервера и сервера Nginx для 100 клиентов.

Таблица 2 – Время прохождения нагрузочного тестирования для 100 клиентов

Количество запросов	Время прохождения тестирования разработанного сервера, с	Время прохождения тестирования сервера Nginx, с
100	0.023	0.039
500	0.085	0.082
1000	0.171	0.129
2000	0.307	0.213
3000	0.383	0.305
4000	0.467	0.401
5000	0.625	0.494

Вывод

Из приведенных таблиц и графиков видно, что в среднем разработанный сервер работает на 30% медленее сервера Nginx. При малом количество клиентов и запросов разница производительности несущественна, однако при числе запросов равному 5000 Nginx работает в 1.5 раза быстрее.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы была достигнута поставленная цель: реализован статический сервер для отдачи файлов с диска на основе пула потоков и мультиплексора poll.

При этом были решены все поставленные задачи:

— проведен анализ предметной области,

— спроектирован алгоритмы работы сервера,

— реализована программу,

— проведено нагрузочное тестирование.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Fielding R. T.*, *Nottingham M.*, *Reschke J.* HTTP Semantics. — 06.2022. — RFC 9110.
2. Hypertext Transfer Protocol – HTTP/1.1 / H. Nielsen [и др.]. — 06.1999. — RFC 2616.
3. *Garg R. P.*, *Sharapov I.* Techniques for Optimizing Applications: High Performance Computing. — USA : Prentice Hall PTR, 2001.
4. *Ling Y.*, *Mullen T.*, *Lin X.* Analysis of optimal thread pool size // SIGOPS Oper. Syst. Rev. — New York, NY, USA, 2000. — Апр. — Т. 34, № 2. — С. 42—55.
5. *Kerrisk M.* Alternative I/O models // The linux programming interface a linux und unix system programming handbook Michael Kerrisk. — No Starch Press, 2018. — С. 1325—1340.
6. Apple M2. — Режим доступа: `https://www.notebookcheck.net/Apple-M2-Processor-Benchmarks-and-Specs.632312.0.html` (дата обращения: 17.12.2024).
7. ab - Apache HTTP server benchmarking tool. — Режим доступа: `https://httpd.apache.org/docs/2.4/programs/ab.html` (дата обращения: 17.12.2024).
8. nginx. — Режим доступа: `https://nginx.org/en/` (дата обращения: 17.12.2024).

ПРИЛОЖЕНИЕ А

Презентация к курсовой работе

Презентация состоит из 8 слайдов.